Under consideration for publication in Math. Struct. in Comp. Science

Formalizing Overlap Algebras in Matita

CLAUDIO SACERDOTI COEN[1†] and ENRICO TASSI[2 †]

[1] *Dipartimento di Scienze dell'Informazione, Università di Bologna via Mura Anteo Zamboni 7, 40127, Bologna (IT)*
[2] *Microsoft Research-INRIA Joint Center, Parc Orsay Université, 28, rue Jean Rostand, 91893, Orsay (FR)*

Received June 2010

We describe some formal topological results, formalized in Matita 1/2, presented in predicative intuitionistic logic and in terms of Overlap Algebras.
Overlap Algebras are new algebraic structures designed to ease reasoning about subsets in an algebraic way within intuitionistic logic. We find that they also ease the formalization of formal topological results in an interactive theorem prover.
Our main result is the existence of a functor between two categories of 'generalized topological spaces', one with points (Basic Pairs) and the other point-free (Basic Topologies).
The reported formalization is part as a wider scientific collaboration with the inventor of the theory, Giovanni Sambin. His goal is to verify in what sense, and with what difficulties, his theory is 'implementable'. We check that all intermediate constructions respect the stringent size requirements imposed by predicative logic. The formalization is quite unusual, since it has to make explicit size information that is often hidden.
We found that the version of Matita used for the formalization was largely inappropriate. The formalization drove several major improvements of Matita that will be integrated in the next major release (Matita 1.0). We show some motivating examples for these improvements, taken directly from the formalization. We also describe a possibly sub-optimal solution in Matita 1/2, exploitable in other similar systems. We briefly discuss a better solution available in Matita 1.0.

1. Introduction

This paper concerns the formalization of formal topology. In particular, we study its algebraic presentation in terms of *Overlap Algebras.*

Overlap Algebras are a new algebraic structure designed for reasoning about subsets in intuitionistic logic. From the perspective of formal mathematics, Overlap Algebras bring great benefits. They both improve formal proofs in terms of readability and also in terms of ease of development. Additionally, the time needed by an interactive theorem prover to check proofs carried out in this setting is sensibly lower.

† This work was partially supported by project CPDA073414 (2008-2010), 'Formalizzazione nel theorem prover interattivo Matita di risultati di topologia formale' of the University of Padova.

Our main result is a formal proof of the existence in predicative intuitionistic logic of a functor between the categories of Basic Pairs and Basic Topologies (Sambin 2011). These are both 'generalized topological spaces', respectively with and without points. The proof, which was novel at the time of the formalization, passes through the existence of one functor between the corresponding algebraic categories, defined in terms of Overlap Algebras, and that of other functors linking the concrete and algebraic categories.

An interesting peculiarity of our work is that the formalization was commissioned by Sambin, the author of the mathematical theory under examination. He was seeking an 'empirical' argument that his mathematics was 'implementable' without major changes within an interactive theorem prover. Another peculiarity is the logic at the base of this theory. It is predicative, and several theorems and constructions state the existence of some mathematical structure *of a given size.*

Matita, an interactive theorem prover, was the system we used for the formalization. Matita has been used in large formalizations of classical number theory (Asperti and Armentano 2008) and intuitionistic abstract integration theory (Sacerdoti Coen and Tassi 2008). As a result, it had become tailored to those formalizations.

We use Matita 1/2 (Asperti et al. , 2007) for the formalization presented here. This system version seemed inappropriate for several tasks, and our work suggested several improvements that have been integrated in Matita 1.0, the next major release.

We sketch the formalization paying particular attention to the troublesome parts, both in terms of limitations of Matita and the inadequacy of the logical foundations that the system is based on. We propose various solutions to the problems identified and describe how they have influenced the design and implementation of Matita 1.0.

The most notable features of Matita 1.0 motivated by this formalization are unification hints (Asperti et al. 2009b); non uniform coercions (Sacerdoti Coen and Tassi 2010); explicit checked universes in the style of (Courant 2002); and separated hierarchies for predicative propositions and data types allowing a controlled use of the type theoretic Axiom of Choice as in (Maietti and Sambin 2005).

The rest of this section is dedicated to a review of the objectives of the formalization. Then we give a 'bird-eye view' of the mathematical background necessary for the formalization. We finish with a summary of the remainder of the paper.

1.1. *Aims and Objectives*

Through the formalization, we aimed to achieve the following orthogonal aims:

1 Verify some of the *proofs* found in Sambin's book (Sambin 2011), not the validity of their statements. No major discrepancy could be tolerated between paper and formal proofs; specifically, we use no automation in the formal proofs. Nevertheless, all the proofs in Sambin's book have an algebraic flavor. This has been fully exploited by re-casting the proofs in an algebraic setting.

2 Provide the first available library containing all the basic notions used in intuitionistic formal topology (reductions, saturations, Overlap Algebras, categories and functors, Basic Topologies, and even some notions not used in the proof presented here like

inductively generated topologies). This library will be used in a forthcoming national project on formalizing formal topology and related computability issues.

3 Demonstrate the feasibility of 'doing' mathematics intuitionistically and predicatively in the context of Overlap Algebras. Overlap Algebras are to intuitionistic subset theory what Boolean lattices are to classical subset theory. We note that the common belief that Heyting algebras play this role is flawed, since in an Heyting algebra we cannot really express the existential quantifier in a positive way. In particular, every Overlap Algebra is an Heyting algebra, but the converse does not hold.

4 Test a new technique to work naturally and succinctly with (non-dependent) setoids in type theory where no explicit support is provided by the system. The technique is optimal in terms of control and efficiency, since no search space is explored at all at the meta level. However, this approach suffers from scalability problems. These problems are solved in Matita 1.0 by unification hints, finding *a posteriori* a generalization of Sozeau's technique (Sozeau and Oury 2008).

5 Push forward the development of Matita 1.0. Several new mechanisms, such as ranked non-coherent coercions, unification hints and non uniform coercions are designed to complete the formalization or to improve it in the future.

1.2. *Mathematical Background*

Classically, a topological space is a pair (X, O) where X is a set of points and O is a set of subsets of X (the open sets). In addition, O satisfies certain well known properties. The usual topological notions, like those of closed set, continuous function, interior, and closure are all given by means of points. Closed sets are simply defined as the complement of open sets, with the definition justified by classical logic.

Classical logic is used extensively in classical topology. The Axiom of Choice is often required in classical point-set topology to recover the points of some construction. For example, Tychonoff's theorem requires the Axiom of Choice to describe the points of the compact topological space obtained as the product of compact topological spaces.

The open sets of O form a lattice. It was noticed (Johnstone 1983) that topology can be formulated in a point-free way by algebraizing this lattice. This allows one to prove most theorems (e.g. Tychnoff's) without the use of the Axiom of Choice. The theory obtained in this way is still based on classical logic, but most of it can be carried out in any topos. Equivalently, it is a generalization of topology that can be done in impredicative logic.

Formal topology began in the '80s as the study of point-free topology done in an *intuitionistic* and *predicative* logic. In a predicative setting the collection of all subsets of a given set is not a set, but a proper class. This has two main consequences. Firstly, we need to explicitly work with bases (when they are sets) and represent open sets as collection of basic opens (the elements of the base). Secondly, several non-constructive arguments that were used, either by means of classical logic (in the classical setting), or by taking the *sup* of a collection of sets with some property (in impredicative point-free topology) now fail. We therefore obtain a more effective theory at the price of carrying more explicit information and duplicating several notions (like that of open and closed collections, that are now given independently). These new notions are informative and reveal many hidden dualities and connections with the purely logical constructions.

Formal topologies are generalization of topological spaces in the sense that the basic opens of a formal topology may not correspond to the open sets of some topological space. Indeed, generally, the collection of points of a formal topology form a class, not a set. The same phenomenon also occurs predicatively. However, it is clear that this remains the motivating example for formal topologies. Furthermore, it is interesting to analyze the situation obtained in intuitionistic and predicative logic, where both the points and the basic opens are explicitly given.

Sambin's Basic Picture is the study of a generalization of this intuitionistic-predicative investigation, obtained by studying two sets linked by a relation. Intuitively these are the set of points, the set of basic opens, and the membership relation. Sambin noted that, even if no other requirement is placed on the two sets and the relation, it becomes possible, by means of purely logical operations, to induce on both sets a saturation and a reduction operator that capture the notion of concrete/formal interior/closure operators. Starting from these operators, it is possible to define continuous relations—a generalization of continuous functions—and many other topological notions.

What is obtained is not another presentation of topology, but a generalization of topology along several axes. For instance, open sets and closed sets do not determine each other. Further, convergence, a fundamental property of topological spaces, is not required from the beginning and can be added on demand. Moreover, these generalizations are preserved even if we add back classical logic and impredicativity. Finally, formal topology (without convergence) may be retrieved if we forget the concrete set (of points) and only remember the formal set (of basic opens). At present, most of the theory of the Basic Picture is unpublished. The main reference will be the monograph (Sambin 2011).

This paper presents the formal verification of one of the aforementioned results in the Basic Picture, establishing the connection between Basic Pairs (the two sets and the relation) and Basic Topologies (formal topologies without convergence). The result can be stated categorically in the following terms: we show the existence of two categories, of Basic Pairs and of Basic Topologies, and the existence of an embedding of the former into the latter. The embedding is full and faithful—unknown when we started the formalization.

Sambin proves the result working on the concrete representations of Basic Pairs and Basic Topologies, i.e. sets, collections of sets and relations between sets. Our first attempt at a formal proof was to closely follow that proof. However, it quickly turned out that working with the concrete definitions and proofs, though feasible, was problematic. We will further discuss this point in Section 3.

Sambin suggests in the same monograph an alternative, more algebraic approach to the study of the Basic Picture that passes through the notion of *Overlap Algebra*. This is the approach we took in our development.

An Overlap Algebra is to intuitionistic subset-theory what a Boolean algebra is to classical subset-theory: an algebraic representation of the 'lattice' of subsets of a given sets. An Overlap Algebra adds to the lattice notions of order *inf* and *sup* a new symmetric binary predicate $\gtrless$. This latter connective asserts, in the intuitionistic sense, the existence of a point in the intersection of two sets. Note that its best approximation in the language of lattice theory is that the intersection of the sets is not empty. The two notions are logically equivalent only assuming classical logic.

Using the notion of Overlap Algebra we can define algebraic versions of Basic Pairs (called *O-Basic Pairs*) and Basic Topologies (called *O-Basic Topologies*). Moreover, we can form the categories of O-Basic Pairs (O-BP) and O-Basic Topologies (O-BTop) and give an embedding of the former into the latter. The embedding is faithful, but not full in general, unless we assume impredicativity. In other words, by dropping the concreteness of an O-Basic Pair we are losing information that cannot be effectively recovered. This is possible if we know that the O-Basic Pair is obtained from a Basic Pair, and this holds under weak assumptions (atomicity) that were not known at the time of our formalization.

Figure 1 summarizes all the categories that form the Basic Picture, where arrows are functors. Those in the upper plane are algebraizing those on the lower plane, and the vertical arrows are full and faithful functors. Those on the rightmost face of the cube will not be considered here (even if partially formalized): they are obtained from those on the leftmost face by adding the hypothesis of convergence, recovering the full theory of topological spaces. Relations (objects of REL) and Overlap Algebras (objects of OA) are basic building blocks of all other categories. In the formalization we define all categories but the convergent ones and all relative functors. The functor from BP to BTop is obtained by composition of those from BP to O-BP and from O-BP to O-BTop and by 'reversing' the one from BTop to O-BTop by means of reduction. The latter point will be made clear once we explain how the various categories are obtained.

$$\begin{tikzcd}
 & O\text{-}BTop & & O\text{-}FTop \arrow[ll] & \\
OA & & O\text{-}BP \arrow[ul] \arrow[ll, dotted] & & O\text{-}CS \arrow[ul] \arrow[ll] \\
 & BTop \arrow[uu] & & FTop \arrow[uu, dashed] \arrow[ll, dashed] & \\
REL \arrow[uu] & & BP \arrow[uu] \arrow[ul] \arrow[ll, dotted] & & CS \arrow[uu] \arrow[ul, dashed] \arrow[ll]
\end{tikzcd}$$

Fig. 1. The Big Picture (Sambin 2011).

The rest of the paper is organized as follows. In Section 2 we talk about the difficulties faced in the formalization and the main techniques adopted. In Section 3 we describe the concrete categories of the formalization, followed by the statement of the main result in Section 4. Section 5 presents the corresponding algebraic categories together with the main advantages of working in this setting, and the proof of the main result is given in Section 6. We close the paper with some conclusions and ideas for future work.

2. Infrastructure

This section introduces the difficulties faced in the formalization. We also discuss the adopted work arounds, addressing shortcomings in the system. These drove the design and development of Matita 1.0.

We will assume the reader has some acquaintance with dependent type theory, possibly with CIC (Coquand and Huet 1988; Paulin-Mohring 1996), and with the proof language of Matita (or Coq, being 'morally' identical).

We write $?_X$ for an implicit (typed) term that has to be inferred by the system. We attach notations to definitions with the following pseudo-command (the actual notation command of Matita is considerably more verbose to allow the definition of multiple interpretations for the same symbol (Padovani and Zacchiroli 2006)):

```
notation "A ⇒ B" := mydef A B.
```

We use ... to avoid repetitions and to routine details. In particular, we will hide the proofs of most theorems and well-formedness conditions, as well as the statements of several lemmata. The complete formalization can be found on-line at the following address: `http://matita.cs.unibo.it/library/formal_topology/`

2.1. *Predicativity and Universes*

Matita 1/2's predicative universe hierarchy was borrowed from Coq 8.0. This hierarchy is an acyclic graph of 'floating' sorts (called universes) for data types.

In set theoretical terms, a universe is the measure of the size of the data type. This idea can be made precise using set theoretical models where the data types in the i-th universe are interpreted with inhabitants of the i-th inaccessible cardinal (Werner 1997).

In particular, we need two universes to speak of proper classes as first class objects; larger classes require more universes. The acyclic graph of universes, inducing a non-linear ordering on sizes, is a method of avoiding explicitly numbering the sizes of data types. Using a graph, we can always interpolate one universe between two given ones, obtaining intermediate sized data types. Further, we can delay the serialization of the sizes of two unrelated data types until we use one of them as a container for the other.

The kernel of the system (i.e. the component checking the correctness of proofs) performs universes inference, allowing the user to use the same ambiguous syntax (Type) for any $Type_i$ in the graph. The exact sizes of data types are invisible to the user, both in input and in output. The constructions made by the user induce the arcs of the graph as size constraints over data types. For instance, making a category of all small sets forces the size of the objects of the category to be strictly larger than the size of sets. If later on the set of all ordinals is formed, the system understands that our sets were really class sized and that categories were larger. As long as the user uses all data types coherently the graph will remain acyclic and remain invisible to the user.

Incoherent use of data types, however, yields wrong arcs in the graph. These lead to cycles only later in the development. The error may be detected checking a construction unrelated to the real mistake, making error localization extremely difficult.

Formal topologists *do* care about size, and their theorems talk about the smallness of constructions, asserting (for example) that some collection is a set and not a proper class. Moreover, predicative logic allows only for quantification over small collections. This clashes with the implicit management of Matita's universe hierarchy in two ways:

— The system may automatically lift a construction meant to be small in response to a mistake (a misuse of the concept) without even telling the user.

— Interactive theorem provers work under the assumption that, if we agree on definitions and theorems statements, the correctness of the proof is granted by the system. Statements are thus the contract between the user and the system and the liberal abuse of Type to mention any sort forces these contracts to be imprecise from the formal topologist's perspective.

In light of these considerations, we must find a solution for two issues. Firstly, we need to allow explicit sized levels in every definition in order to keep constructions under control. Secondly, we need to provide equivalent notions that differ only in size.

The first problem is impossible to solve without changing the kernel itself (we do this in Matita 1.0), but a reasonable approximation can be found. In place of using a fresh Type everywhere, at the beginning of the development we named some universes and we use only those names throughout. Moreover, we immediately force a linear ordering on the named universes by typing each one with the next:

```
definition Type4 : Type := Type.     definition CProp4 := Type4.
definition Type3 : Type4 := Type.    definition CProp3 := Type3.
...
definition Type0 : Type1 := Type.    definition CProp0 := Type0.
```

In the style of (Cruz-Filipe et al. 2004), we assign two names to each universe: Type_i and CProp_i. We will inhabit the first with data types and the second with predicative propositions. As predicative propositions are the same as data types, no typing rule prevents eliminating a proposition to obtain a data type. This implicitly assumes that we are working in a theory validating the type theoretic Axiom of Choice, diverging from the logic of Sambin (Maietti and Sambin 2005). This issue can only be solved by equipping the type theory with two distinct predicative hierarchies with different elimination rules.

Aside from admitting the Axiom of Choice, there are other reasons that make the solution described here not fully correct. Firstly, nothing suggests that Type0 will remain the first type of the hierarchy and avoid any raising. Similarly, nothing forces Type1 to be the immediate successor of Type0. Nevertheless, hiding a construction explicitly declared in Type1 inside one declared in Type0 is explicitly forbidden. In our experience, all inconsistent uses of data types have been immediately detected by Matita.

We resort to manual code duplication to provide different notions that are equivalent up to size without introducing in the system universal polymorphism. We keep duplication under control using orthogonal definitions and coercions as detailed in Subsection 2.3.

2.2. *Rewriting with setoids*

The constructive flavour of the mathematics being formalized implied special care was needed in treating equality by avoiding any form of quotienting. We work with Bishop's sets, also called setoids.

Setoids are types, meant to be a representation of data. They come equipped with an equivalence relation, representing the requisite notion of equality.

```
record equivalence_relation (A : Type0) : Type1 :={
   eq_rel :2> A → A → CProp0;
   refl : reflexive A eq_rel; sym: symmetric A eq_rel; trans: transitive A eq_rel }.
record equivalence_relation1 (A : Type1) : Type2 :={ ...
notation "A = B" := (eq_rel ? A B).   notation "A = B" := (eq_rel1 ? A B).
...
record setoid : Type1 :={ carr:> Type0; eq: equivalence_relation carr }.
record setoid1 : Type2 :={ carr1:> Type1; eq1: equivalence_relation1 carr1 }.
```

Note that we overload the same notation for small and large equivalence relations. This is handled by a distinctive feature of Matita, where overloaded notation is admitted by explicitly interacting with the user in case of ambiguities that are not resolved by the context (Sacerdoti Coen and Zacchiroli 2008). While this feature was exploited at the beginning almost everywhere in the formalization, we later decided to often use more explicit notations in the script—for instance to manifest the size of some notions—for the sake of mathematicians reading the script.

The ? symbol stands for a wildcard. In this particular case it stands for the equivalence relation that the eq_rel projection is applied to.

Bishop's functions between data types (also called morphisms of setoids) are functions on the representations that respect the intended equality. In order to replace a term with an equivalent one in some context, i.e. to perform a rewriting step, the context must respect the equivalence relation. This is true for all contexts entirely made of morphism applications. However, the latter is a statement at the meta-level and, every time a rewriting takes place, the user needs to provide an explicit proof that the context has the required property. Without some automation from the system, working with setoids is particularly tedious.

Systems like Coq provide high-level tactics to automate this task. Intuitively, a proof is built by collecting all the functions used in the context, looking up each one of them in a user-provided database of proofs of compatibility with equivalence relations, and finally composing these proofs together. In (Sacerdoti Coen 2006) the search is implemented in an *ad hoc* way, but the correctness of the procedure is granted using reflection.

Later, Coq was equipped with a more flexible implementation (Sozeau 2009). This replaced the *ad hoc* search with the automatic search of type classes instances. The idea is to build from the context structure a term made of missing information that, when inferred, triggers the inhabitation of the type class.

In Matita 1/2 we have no support at all for rewriting in the context of setoids. Nevertheless, we have been able to provide the user with lightweight setoid rewriting using the following technique, tested on this formalization for the first time.

Intuitively, we represent every morphism as a record that packs a function along with the property of being a morphism. We declare a coercion (the first record projection, typed with ':1>' or ':2>' instead of ':') from the morphism to the packed function. We present both the declaration of unary and binary morphisms, but we will here talk about unary morphisms only. We however note here that currying binary morphisms is problematic, and not done, as we will explain later.

```
record unary_morphism (A, B : setoid) : Type0 :={
   fun1:1> A → B; prop1: ∀a,a'. a = a' → fun1 a = fun1 a' }.
notation "A ⇒ B" := unary_morphism A B.
record binary_morphism (A, B, C : setoid) : Type0 :={
   fun2:2> A → B → C; prop2: ∀a,a',b,b'. a = a' → b = b' → fun2 a b = fun2 a' b' }.
notation "A × B ⇒ C" := binary_morphism A B C.
record unary_morphism1 (A, B : setoid1) : Type1 :={ ...
notation "A ⇒1 B" := unary_morphism1 A B.
record binary_morphism1 (A, B, C : setoid1) : Type1 :={ ...
notation "A × B ⇒1 C" := binary_morphism1 A B C.
```

Every time the user inputs a morphism applied to something, the system automatically inserts the coercion into the term and hides it from the user. The resulting term looks the same, but it carries much more information. In particular, all proofs that functions occurring in a term respect the appropriate equalities are kept inside the term. Therefore, no complex meta-level search mechanism needs to be implemented. Instead, these proofs can be easily recovered by standard unification in the following way.

Suppose that we want to replace E_1 with E_2 in (P (f E_1)), where P and f are unary morphisms, P yields a proposition and we know that $E_1 = E_2$ (hypothesis H). Under the 'notational bonnet', the term is represented as (fun21 ?? P (fun1 ?? f E_1)) (where fun21 and prop21 are the names of the two fields of a large unary morphism, like the ones that return propositions, and ? represent an implicit argument that the system is supposed to automatically infer from the context). In order to inhabit (P (f E_2)), it is sufficient to consider the term (fi ?? (prop21 ?? P ?? (prop1 ?? f ?? H))) where all questions marks are inferred by the system and fi is a proof that $\forall$P,Q. P $\iff$ Q $\to$ (Q $\to$P). This is true, since H has type $E_1 = E_2$, (prop1 ?? f ?? H) has type f E_1 = f E_2 and (prop21 ?? P ?? (prop1 ?? f ?? H)) has type P (f E_1) = P (f E_2). Since the intended equality on propositions is co-implication, the whole term has type P (f E_2) $\to$ P (f E_1) that, applied to the current goal, turns (P (f E_1)) into (P (f E_2)) as required.

Now, the interesting thing is to look at the same proof term once we erase all references to P and f. The remaining term (fi ?? (prop21 ?? $?_P$?? (prop1 ?? $?_f$?? H))) has type:

```
fun21 ?_A CProp0 ?_P (fun1 T ?_A ?_f E_2) →fun21 ?_A CProp0 ?_P (fun1 T ?_A ?_f E_1)
```

or, hiding all coercions, $?_P$ ($?_f$ E_2) $\to ?_P$ ($?_f$ E_1). I.e. the term is a universal proof to rewrite E_1 into E_2 in any applicative context of the form $?_P$ ($?_f$ _). However, the universal term is still cumbersome to write. This can be overcome by introducing *ad hoc* infix notations for prop1, prop21, if and related operators, and by overloading them at every size using Matita's ambiguous notation facility (Sacerdoti Coen and Zacchiroli 2008):

Notation	Definitions		
"† c"	prop1 ????? c	prop21 ????? c	...
"l ‡ r"	prop2 ???????? l r	prop22 ???????? l r	...
". t"	if ?? t		
"#"	refl ???	refl2 ???	...
"t^{-1}"	sym ??? t	sym2 ??? t	...
".= t"	trans ???? t	trans2 ???? t	...

Thanks to this notation, the universal term to rewrite E_1 into E_2 using H in any context ($?_P$ ($?_f$ E_1)) is simply (. † († H)) to be read as: *replace* (the dot operator) *under a unary operator* (†)*, then under another unary operator* (†) *using H.* Hence the universal term is simply a path in the expression that specifies where rewriting needs to occur. A proof of reflexivity # is used to mark a place where rewriting does not occur; symmetry (H^{-1}) to rewrite in opposite direction; transitivity .= to rewrite in the left hand side of an equation. Parallel rewriting is obtained by using more than one hypothesis in the path.

The following examples should convince the reader of the flexibility of the approach, and the ease of writing down, even by hand, and reading back the rewriting paths. Of course, it is also easy to automatically generate them, for example by clicking with the mouse on sub-terms, as we already do in Matita for standard rewriting patterns.

Expression	Universal term	Result
P(a + (b ∗ f (E_1)))	. †(# ‡ (# ‡ († H)))	P(a + (b ∗ f(E_2)))
a ∗ c + E_2 = E_1 + b	.= # ‡ H^{-1}	a ∗ c + E_1 = E_1 + b
E_1 + E_2 ∗ E_1 = 0	.= H‡ (H^{-1} ‡ #)	E_2 + E_1 ∗ E_1 = 0

Setoid rewriting, expressed in this form, is pervasive in all the proofs of the present formalization. In particular, once we enter the algebraic setting of overlap algebras, most proofs are 'diagram chasing'. This style of reasoning corresponds to long chains of rewriting steps, each one performed by a rewriting path.

The main drawback of the above technique is that it forces all terms to be written at the setoid level, yielding larger proof terms. Moreover, one must be careful to avoid uncontrolled reduction of terms by the system, that could simplify away the rich structures. On the other hand, no complex look-up of properties in tables needs to be performed, script interpretation is very fast, the equalities to be used are never ambiguous, making script interpretation deterministic, and we did not have to change the system in any way. Finally, in Matita 1.0, it will be possible to combine our technique with unification hints (Asperti et al. 2009b) so terms can be written normally (i.e. not like morphism applications). Then, the unifier must work out how to enrich them into morphisms, in the spirit of (Sozeau 2009). The following example should clarify this idea.

Suppose the current goal is y ∈ {x} where we no longer hide in the notation the application of morphisms, so that ∈ is really a function and not a morphism and, removing the notational sugar, we obtain (mem y (singleton x)). If we have a proof H that $y =_A x$, we can build the universal proof term (. H‡#) as before whose type is:

```
fun2 A ?_B' ?_C ?_f x ?_z →fun2 A ?_B' ?_C ?_f y ?_z
```

Applying the universal term, we trigger the unification problem:

```
mem y (single x) = fun2 A ?_B' ?_C ?_f y ?_z
```

This is no longer immediately solved by the unifier. In particular, unifying (single x) with $?_z$ is easy, but completing the unification means finding a record instance $?_f$ such that its fun2 projection is mem. Even if the user may have already shown that mem is a morphism—providing a proof term that is the sought instance for $?_f$—the unifier does not know about the existence of that proof term and its applicability. Thus, to solve the unification problem, we need to 'hint' the solution to the unifier in advance once and for all. This is done in Matita 1.0 by means of unification hints.

2.3. *Category theory in a predicative setting*

2.3.1. *A matter of size.* Traditionally, categories are defined as a *collection* of objects, a *collection* of arrows (or morphisms), two maps associating to each arrow its domain and codomain (two objects) and a partial associative operation (composition) over compatible morphisms that has neutral elements, satisfying some basic properties. Alternatively, instead of a single collection of arrows and two maps, we can assume for every pair of objects a *collection* of arrows between them (misleadingly called the Hom-Set collection).

For most interesting categories (like SET, whose objects are sets and whose arrows are functions), the collections are not sets, but proper classes. For instance, the collection of objects of SET is actually a proper class. In these cases the categories are called *large*. It is often the case that a category is large, but the Hom-Sets are all sets. In this case the category in question is called *locally small*. SET is an example of a locally small category.

Since we must handle proper classes as first class objects, standard set theory (ZFC) is not sufficient and, classically, we need more complex set theories like Von Neumann-Bernays-Goëdel (NBG) set theory whose ontology is made of sets (that can have and can be members) and proper classes (that cannot be members).

Predicative logic further complicates the scenario in such a way that even NBG set theory is no longer expressive enough to define some useful categories. Indeed, in a predicative setting, the collection of all subsets of a set no longer forms a set, but it is a proper class, and a collection of collections of subsets (like a category of lattices of subsets) may turn out to have objects that do not even form a proper class. For instance, the category REL (whose objects are sets and whose arrows are binary relations) that is impredicatively locally small is only large predicatively, and the category OA (that we will introduce later) needs to be very large since it has as a full sub-category a category whose objects are the collections of all subsets of some set. Hence, when reasoning predicatively in set theory, we need a theory that has an extremely rich ontology, like one with a full Von Neumann hierarchy of sets, that is roughly equivalent to the assumption of the existence of a tower of inaccessible cardinals.

We already know Matita's type theory is characterized by a hierarchy of cumulative universes. It transpires that (Werner 1997) the set theoretical expressive power of the theory with n universes and the Axioms of Excluded Middle and Choice is equivalent to ZFC with n inaccessible cardinals. Hence, we are theoretically able to define very large categories (it will turn out that we only need small, large and very large categories in the present formalization).

However, there are practical problems in concretely doing that. Informally a mathematician would define categories as very large ones (i.e. categories whose collections are very large classes) and will reason with assumptions of the form '*the Hom-Set are sets*' to reduce the size of known categories. This kind of judgment cannot be expressed in this form in type theory. More precisely, systems like Matita implement a syntax-directed typing judgment that associates to each type expression T a unique inferred 'principal kind' Type_i. The following rules allow us to use T as a member of Type_j for each $j \geq i$, but not as a member of Type_l with $l < i$.

$$\frac{i \leq j}{\Gamma \vdash \mathrm{Type_i} \leqq \mathrm{Type_j}} \qquad \frac{\Gamma \vdash f : \Pi y : A.B \qquad \Gamma \vdash x : A' \qquad A' \leqq A}{\Gamma \vdash f\ x : B[x/y]}$$

Statements like ($\mathrm{T} : \mathrm{Type}_l$) (i.e. casting of T to the kind Type_l, that can be thought as (λX: Type_l.X) T), are rejected by the system even if T (or better, one of its reducts) has 'principal kind' Type_l. The previous remark can be rephrased as follows: given T_1: Type_i and T_2: Type_j, it may be the case that T_1 and T_2 are convertible, but T_1: Type_j or T_2: Type_i may not hold. For example **N** has principal kind Type_0 while (λX: Type_1.X) **N** has principal kind Type_1.

We try to avoid the problem by adopting a redundant coding of categories: instead of giving a single definition of a (very large) category, we give several definitions, one for each size. Note, however, that we have two collections in each category. Hence we need to define n^2 categories where n is the largest size we need for a collection, and we need to develop n^2 morally identical theories (up to size quantifications). Moreover, if we allow functors between categories with different sizes, we obtain n^4 theories, and so on.

In our formalization in Matita 1/2 we keep code duplication under control by artificially considering only n-sized categories where the collection of objects and each Hom-Set have size n, and we cope with the 'principal kind' problem in case of locally n-small categories. Similarly, we only consider functors between equally sized categories, even if most of the functors we consider will be from smaller to larger categories, requiring us to cope once again with the problem.

Another inconvenience is that CIC has cumulativity rules for kinds only. For example an argument of kind Type_0 can be passed to function expecting a Type_1, but the same does not hold for Σ-types (coded as inductive types). As a consequence a small category can not be used as a large one transparently, as in (Luo 1989).

As a workaround we have a function that disassembles a small category, copies its components, and reassembles them into a large category. The insertion of these lifting functions in appropriate places can be delegated to the system by means of the *coercion* mechanism: every time a type error occurs, i.e. some term has inferred type S but is expected to have type T, the system can automatically insert around the term an application of a user-provided function from S to T. In our case, S and T would be respectively small and large structures.

The problem with this approach is that, once the insertion of the lifting function is delegated to the system, the system will compose it with the insertion of other delegated functions, like the one that extract the carrier from a structure. Hence the system will end up in situations where it could lift some structure to a larger size, as an intermediate step hidden to the user, and, moreover, it could obtain multiple incoherent ways of promoting a term that differs on the size of the intermediate constructions. To avoid this issue, we introduce in Matita 1/2 the possibility to manually rank automatically generated coercions between two types, in order to make the system prefer coercions that do not pass through intermediate large structures.

2.4. *Categories and setoids*

A setoid, or Bishop's set, is a type equipped with an equivalence relation. This relation is meant to be the intended notion of equality over objects of that kind. A morphism between setoids is a function that respects the equivalence relation, mapping equivalent inputs to equivalent outputs. Setoids remove the need to work with quotients: in place of quotienting some type to obtain a different one, we may change its equivalence relation.

Dependent setoids are types that are defined in a context of variables that range over setoids. A dependent setoid $T(x)$ is a dependent type T over x equipped with a function that maps elements of $T(x)$ to elements of $T(y)$ for all y equivalent to x. Dependent setoids largely complicate the technicalities involved in the use of setoids, since two terms inhabiting respectively $T(x)$ and $T(y)$ for equivalent but different x and y, have different types and thus they cannot even be compared without converting one of them from $T(x)$ to $T(y)$ beforehand. Hence there is a long tradition in type theory of avoiding dependent setoids as much as possible.

In the current theory we follow the same approach with the following choice: the collection of objects of a category is a n-sized type (or, equivalently, a setoid whose equivalence relation is Leibniz equality), while the Hom-Sets are (non dependent) setoids. This suffices for a large class of examples of categories, and it is in the spirit of 2-categories. From the purely categorical perspective, we are just working with P-categories (Cubric et al. 1997) that are PER-enriched categories.

2.5. *The formal definition*

As an example, we present the definition of a large (1-sized) category. The corresponding definition for small (0-sized) and very large (2-sized) categories is obtained syntactically by decrementing/incrementing each integer in the definition.

```
record category : Type1 :={
   objs:> Type0; arrows: objs → objs → setoid;  id : ∀o:objs. arrows o o;
   comp: ∀o1,o2,o3. (arrows o1 o2) × (arrows o2 o3) ⇒ (arrows o1 o3);
   comp_assoc: ∀o1,o2,o3,o4.∀a12:arrows o1 ?.∀a23:arrows o2 ?.∀a34:arrows o3 o4.
      (a12 ∘ a23) ∘ a34 = a12 ∘ (a23 ∘ a34);
    id_neutral_left  : ∀o1,o2. ∀a: arrows o1 o2. (id o1) ∘ a = a;
    id_neutral_right : ∀o1,o2. ∀a: arrows o1 o2. a ∘ (id o2) = a }.
```

The :> syntax declares the record projection objs as a coercion. It allows the user to write ∀C:category.∀o:C... that, without the automatic insertion of the projection objs around C, would be ill-typed, since the term C, is used as a type in the second quantification.

As discussed in the previous section, objects are not quotiented, but Hom-Sets (here called arrows) are, and their composition is required to be a morphism. The locally 0-small category of sets, whose objects are setoids and whose arrows are unary morphisms is defined as a 1-sized category by lifting every 0-sized Hom-Set (a setoid) to be 1-sized using setoid1_of_setoid .

```
definition SET: category1.
 constructor 1 [ apply setoid | apply (λS,T.setoid1_of_setoid (unary_morphism_setoid S T))
... qed.
```

Since proofs are part of the definition of a category, we give the definition interactively. In other words, when we execute the **definition** SET: category1 command we are left with an open 'proof' obligation that asks the user to inhabit the type category1. The **constructor** 1 command is used to inhabit record types, and it opens a new (dependent) proof obligation for each record field. The first one corresponds to the objs1 field, and requires an inhabitant for Type1, i.e. the type of objects of the category. We provide setoid, the type of all setoids. The second obligation asks for the Hom-Sets, i.e. an inhabitant of ∀S,T:setoid. unary_morphism1 S T and we inhabit it with the small type of morphisms between S and T lifted to large morphisms.

As a second important example, we show the definition of the 1-sized, non-locally 0-small category REL whose morphisms are binary relations over setoids, i.e. 1-sized binary morphisms on the 1-sized setoid of propositions quotiented by equi-provability. Using a well-known notational trick, we declare binary relations using a degenerate record with just one field to associate a particular notation (a best approximation to the infix notation, x ♮R y) to applications of relations.

```
record binary_relation (A,B: SET) : Type1 :={ satisfy:> A × B ⇒_1 CPROP }.
```

We then form a setoid of binary relations by identifying relations extensionally, and build a category out of it.

```
definition  binary_relation_setoid :  SET → SET → setoid1.
 intros (A B); constructor 1; [ apply (binary_relation A B)
  | constructor 1; [ apply (λA,B.λr,r'. ∀x,y. r x y ↔r' x y) ...
notation "x ♮R y" := fun21 ??? (satisfy ?? R) x y.
definition REL: category1.
 constructor 1; [ apply setoid | intros (T T1); apply (binary_relation_setoid  T T1)
...
notation "A  ⇒^r_1 B" := arrows1 REL A B.
```

We also define functors, as follows:

```
record functor (C1: category) (C2: category) : Type1 :={
  map_objs:1> C1 → C2;
  map_arrows: ∀S,T. (arrows2 C1 S T) ⇒ (arrows2 C2 (map_objs S) (map_objs T));
  respects_id : ∀o:C1. map_arrows ?? (id C1 o) = id C2 (map_objs o);
  respects_comp: ∀o1,o2,o3.∀f1:arrows C1 o1 o2.∀f2:arrows C1 o2 o3.
    map_arrows ?? (f2 ∘ f1) = map_arrows ?? f2 ∘ map_arrows ?? f1 }.
notation "F⇒ f" := fun11 ?? (map_arrows ?? F ??) f.
```

Note that a functor can be applied to objects naturally, thanks to the coercion map_objs2, which requires the additional $_\Rightarrow$ suffix to map arrows.

Finally, we introduce the notions of fullness and faithfulness for functors:

```
notation "A ⇒^c_3 B" := arrows3 CAT2 A B.
definition full2 :=λA,B:CAT2.λF:carr3 (A ⇒^c_3 B).∀o1,o2:A.∀f.∃g:arrows2 A o1 o2.F⇒ g =f.
definition faithful2 :=λA,B:CAT2.λF:carr3 (A ⇒^c_3 B).
   ∀o1,o2:A.∀f,g:arrows2 A o1 o2.F⇒ f =F⇒ g →f =g.
```

CAT2 is a very large category whose objects are large categories and whose arrows are large functors.

2.6. *Working with rich structures*

We already motivated the idea of always working with morphisms in place of functions in order to implement setoid rewriting by means of notation. The very same idea can be extended to richer structures: it is sufficient to always replace a poor structure with an enriched one in order to make notation project out the additional structure without performing any proof search. For instance, as soon as the category SET of sets is defined, we always declare unary morphisms as arrows of the category SET in order to have composition for free. So, for instance, we write ∀f: A $\Rightarrow_1$ A (or, without notation, ∀f: arrows1 SET A A) in place of the convertible declaration ∀f: unary_morphism A A so that the notation f ∘ f, associated to the composition field of a category, is accepted by the system and denotes the intended composition function for the morphisms under examination.

In other words, we get the same flexibility of type classes, and the only price we pay is the quantifying and declaration of objects in rich structures. The only real problem of this approach is that it is not stable under future extensions of the library. For instance, if we did not define the category SET, then we would have declared all functions as unary morphisms; if later we decide that composition is also useful, then we would have to introduce SET and re-type every lemma already proved to quantify over objects and arrows of SET.

As for setoids, the introduction in Matita 1.0 of unification hints definitely solves this issue by allowing one to always work with simple structures, and enrich them on-the-fly during unification.

3. Concrete presentation

We now present the various categories that have been formalized, starting from the concrete ones (categories whose objects are built from sets and power-classes).

3.1. *Sets and their operations*

The first construction we formalize is the powerclass, along with the membership operation. We embed the characteristic function of the powerclass in an inductive type to be able to attach a notation to it, and we then prove that it forms a 1-sized setoid when quotiented by standard set theoretic equivalence (i.e. double inclusion):

```
record powerclass_carrier (A: objs1 SET) : Type1 :={ mem_operator: A ⇒₁ CPROP }.
notation "Ω^A" := powerclass_carrier A.
definition subseteq_operator: ∀A:SET. Ω^A → Ω^A → CProp0 :=
 λA:SET.λU,V.∀a:A. a ∈ U → a ∈ V.
notation "U ⊆ V" := subseteq_operator ? U V.
definition powerclass_setoid1: SET → SET1.
 intros (T); constructor 1; [ apply Ωˆ T | ...apply (λU,V. U ⊆ V ∧ V ⊆ U) ...
coercion setoid1_of_SET1 : objs2 SET1 → setoid1 :=λx.x.
```

Following the approach of making constructions as rich as possible we defined our Ω^A as an object (the obj2 projection is automatically inserted around SET since it is not a type, but a term) of the category of classes SET1. The interesting trick behind this definition is the *identity* coercion defined on the last line. The objects of the category of classes are indeed 1-sized setoids but would not be recognized as such by the coercion mechanism, that approximates type comparisons with a syntactic (reduction free) check.

In particular this coercion is necessary to allow Ω^A to be automatically embedded in the 2-size setoid, allowing one to write $\Omega^A \Rightarrow_2$ B. The system knows how to lift the objects of SET1 into a 2-sized setoid: they are first coerced (with a no-op) to inhabitants of a 1-sized setoid, then embedded thanks to setoid2_of_setoid1 into the larger setoid.

The other constructions involving sets are not interesting, and are thus summarized in the following table. The notation {x | P x} hides the term (mk_powerclass_carrier ? (λx.P x)).

Name	Notation	Type	Definition
mem	"a ∈ U"	$\forall$A. A $\times \Omega^A \Rightarrow_1$ CPROP	λx,S.mem_operator A x S
subseteq	"U ⊆ V"	$\forall$A. $\Omega^A \times \Omega^A \Rightarrow_1$ CPROP	λU,V. subseteq_operator A U V
overlaps	"U ≬ V"	$\forall$A. $\Omega^A \times \Omega^A \Rightarrow_1$ CPROP	∃x.x ∈ U ∧ x ∈ V)
intersects	"U ∩ V"	$\forall$A. $\Omega^A \times \Omega^A \Rightarrow_1 \Omega^A$	λU,V. {x | x ∈ U ∧ x ∈ V }
union	"U ∪ V"	$\forall$A. $\Omega^A \times \Omega^A \Rightarrow_1 \Omega^A$	λU,V. {x | x ∈ U ∨ x ∈ V }
singleton	"{ a }"	$\forall$A:setoid. A $\Rightarrow_1 \Omega^A$	λa:A.{b | a = b}
big_intersects	"⋂ f"	$\forall$A,I:SET.(I $\Rightarrow_2 \Omega^A$) $\Rightarrow_2 \Omega^A$	λc.{x | ∀i:I. x ∈ c i}
big_union	"⋃ f"	$\forall$A,I:SET.(I $\Rightarrow_2 \Omega^A$) $\Rightarrow_2 \Omega^A$	λc.{x | ∃i:I. x ∈ c i }

Note that all operators are defined as morphisms $\Rightarrow_1$ or $\Rightarrow_2$ on some large or very large setoids. CPROP is the category whose objects are small propositions quotiented by co-implication. Every definition requires an explicit proof of preservation of equivalence, which is tedious but simple. The formal script is long, thus omitted.

3.2. *Relation images*

One of the most powerful ideas of the Basic Picture is that relations can be characterized by looking at their universal and existential images and counter-images, effectively hiding the use of quantifiers. The formalization of the four images poses no interesting challenges. It will be worth keeping the following concrete definitions in mind whilst reading their algebraic counterparts in the definition of relations between overlap algebras (Section 5.1). The superscript ^{r} on an arrow reminds the reader that the arrow belongs to REL.

Name		Type	Definition
image		$\forall$ U,V. (U $\Rightarrow_1^r$ V) $\times\, \Omega^U \Rightarrow_1 \Omega^V$	λ r,S. {y $\mid$ $\exists$ x:U. x ♮r y $\wedge$ x $\in$ S}
minus_star_image	”r^{-*}”	$\forall$ U,V. (U $\Rightarrow_1^r$ V) $\times\, \Omega^U \Rightarrow_1 \Omega^V$	λ r,S. {y $\mid$ $\forall$ x:U. x ♮r y $\rightarrow$ x $\in$ S}
star_image	”r^{*}”	$\forall$ U,V. (U $\Rightarrow_1^r$ V) $\times\, \Omega^V \Rightarrow_1 \Omega^U$	λ r,S. {x $\mid$ $\forall$ y:V. x ♮r y $\rightarrow$ y $\in$ S}
minus_image	”r^{-}”	$\forall$ U,V. (U $\Rightarrow_1^r$ V) $\times\, \Omega^V \Rightarrow_1 \Omega^U$	λ r,S. {y $\mid$ $\exists$ x:V. x ♮r y $\wedge$ x $\in$ S}

3.3. *Basic Pairs*

The first category BP that we consider from the Basic Picture is that of Basic Pairs. The objects of the category are pairs of setoids (called the *concrete* and the *formal* side) linked by a generic relation that we denote by ⊩ (to be read as ‘forces’ or ‘is in’). The topological intuition is that concrete items are points of some space whose collection of points forms a set (like the set of rational numbers). The formal items are basic open sets, i.e. the elements of a basis of a topology that is 0-sized (like the set of open intervals with coefficients in **Q**).

$x \Vdash a$ means that the point x belongs to the basic open a. However, nothing in the definition forces this interpretation and the two sides can be freely chosen. In that case the intended reading of $x \Vdash a$ is that x enjoys the observable property a. As we will see later, the forcing relation suffices to induce a lattice structure on the formal side. However, nothing forces the induced infimum operator to coincide with intersection, wherever in a topological space the collection of opens form a lattice whose infimum operator is intersection. Also in this sense Basic Pairs are a strict generalization of the notion of topological space.

```
record basic_pair: Type1 :={ concr: REL; form: REL; rel: concr ⇒₁ʳ form }.
notation ”x ⊩ y” := fun21 ??? (rel ?) x y.    notation ”⊩” := rel ?.
```

A morphism of relation pairs is a couple of relations acting respectively on the concrete and formal side that make the diagram, formed by also considering the two forcing relations, commute. The topological intuition is that of a continuous function that maps points to points and counter-maps basic opens to basic opens by respecting the membership relation. Note, however, that the two maps (in opposite direction) are here substituted by two relations, obtaining a genuine generalization of the topological case.

```
record relation_pair (BP1,BP2: basic_pair): Type1 :={
  concr_rel : (concr BP1) ⇒$_1^r$ (concr BP2); form_rel: (form BP1) ⇒$_1^r$ (form BP2);
  commute: ⊩ ∘ concr_rel = form_rel ∘ ⊩ }.
```

What is apparent from the code in Matita is that almost no typing or disambiguation information is provided in the definition, similar to the paper definition of Sambin. This is again possible since all types have been maximally enriched and thus the system already knows that all the arrows involved are relations (arrows of REL) and that their composition and equivalence relations are the ones of REL too. On the other hand we make explicit (almost) all size information (the subscripts 1) to explicitly verify that everything is done in a large, but not very large setting.

To obtain a category we still need to say what we mean by equivalent relation pairs and how to compose them. Composition is composition component-wise. Equivalence is subtler: two relation pairs are equivalent when composed, with forcing, we obtain equivalent relations (in the sense of REL, i.e. extensionally). Topologically, we are identifying two relation pairs if they map every point to 'topologically indistinguishable' ones, i.e. to points that cannot be separated by any basic open. This definition is equivalent to the classical one for T0 spaces.

The formal definition in Matita is the following (we omit the quite long proofs required to form a morphism and a category):

```
definition relation_pair_setoid : basic_pair → basic_pair → setoid1.
 intros; constructor 1; [ apply (relation_pair b b1)
  | intros; constructor 1; [ apply (λ r,r'. ⊩ ∘ r_c = ⊩ ∘ r'_c); ...
lemma relation_pair_composition:
  ∀ o1,o2,o3: basic_pair .
   relation_pair_setoid o1 o2 → relation_pair_setoid o2 o3 → relation_pair_setoid o1 o3.
 intros (o1 o2 o3 r r1); constructor 1; [ apply (r1_c ∘ r_c) | apply (r1_f ∘ r_f)...
definition BP: category1. constructor 1; [ apply basic_pair | apply relation_pair_setoid ...
notation "A ⇒$_1^{bp}$ B" := arrows1 BP A B.
```

Comparing the definition of BP with the traditional definition of the category of topological spaces, we note a totally different definitional style that is clearly reminiscent of point-free topology (even if Basic Pairs talk about points). In particular, relation pairs, their composition, and equality are defined in point-free style using commuting diagrams (a clearly categorical style) that hide the point-wise direct definitions based on alternation of quantifiers. For instance, the commuting diagram that defines equivalence over relation pairs hides the statement $\forall x, x', a, a'\ (\exists y, x r_c y \wedge y \Vdash a \iff \exists y', x' r_c y' \wedge y' \Vdash a')$.

This point-free definitional style is pervasive throughout the Basic Picture and, surprisingly, it has a strong impact on the proofs too. In particular, not only are most of the hypotheses, definitions and statements we are interested in given as commutative diagrams, but the proofs themselves can be obtained simply by 'diagram chasing'. For instance, the following is half of the proof that equivalence of relation pairs can be also characterized looking at the formal side only:

```
lemma eq_to_eq':
 ∀ o1,o2.∀ r,r': relation_pair_setoid o1 o2. r = r' → r_f ∘ ⊩ = r'_f ∘ ⊩ .
 intros 5 (o1 o2 r r' H); change in H with (⊩ ∘ r_c = ⊩ ∘ r'_c);
 apply (.= (commute ?? r)^-1); apply (.= H); apply (commute ?? r').
qed.
```

Declaratively, the proof looks like the following, and it is given by diagram chasing only:

$$\begin{tikzcd}
C_1 \arrow[r, dash, "\Vdash_1"] \arrow[d, dash, shift right, "r'_c"'] \arrow[d, dash, shift left, "r_c"] & F_1 \arrow[d, dash, shift right, "r'_f"'] \arrow[d, dash, shift left, "r_f"] \\
C_2 \arrow[r, dash, "\Vdash_2"] & F_2
\end{tikzcd}$$

$$\begin{aligned}
r_f \circ \Vdash_1 &= \Vdash_2 \circ r_c && \text{by (commute } r)^{-1} \\
&= \Vdash_2 \circ r'_c && \text{by hypothesis H} \\
&= r'_f \circ \Vdash_1 && \text{by commute r'}
\end{aligned}$$

If we make the mistake of expanding the definitions revealing the quantifiers, in place of this very elegant and clear point-free proof we obtain a point-wise proof that is longer and messy. Moreover, in case we want to exploit automation, after expansion of definitions we obtain a search space that is very large. Further, we trade equational reasoning (for which we have automatic efficient semi-decision procedures) for full first order logic (that admits less efficient automation). We conclude that we should not expand definitions when this is not strictly necessary. However, even if we avoid expansion in the proof script, the system is likely to expand definitions automatically when checking convertibility of terms or during unification, with a potential (and sometimes concrete) major impact on type-checking performances and the time required to re-process the proof script. This raises some major questions that our formalization helped to settle:

1 Is it possible to introduce a purely algebraic description of formal topology and the Basic Picture in such a way that concrete complex structures and definitions are replaced by abstract entities and equations (or, more generally, relations)?

2 What algebraic setting is complete enough to hide concrete definitions given in terms of the universal and existential quantifiers of intuitionistic logic retaining the same computational content?

We will come back to these questions in Section 5.1 when we will explicitly introduce an algebraic presentation of Basic Pairs, called O-Basic Pairs, obtained my mimicking the Basic Pairs definitions (and proofs) after having replaced sets and relations (i.e. the category REL) with its algebraic counterpart, the category of Overlap Algebras.

3.4. *The hidden topological structure of Basic Pairs*

Despite the definition of a Basic Pair requiring only an extremely weak structure (just two sets linked by an arbitrary relation), the theory of Basic Pairs is extremely rich. We induce, starting from the relation of the Overlap Algebra, saturations and reduction operators, on both the concrete and formal side. In the topological reading, these operations correspond to the concrete/formal interior and closure operators. For instance, the interior of a set S of concrete points is the set made of all those points that are (according to $\Vdash$) in at least one basic open whose points (according to $\Vdash^{-1}$) are all in S, and an open set is a set equal to its interior. The interior operator is a function on sets of points that is a reduction operator, i.e. an idempotent and monotone operator such that the interior of S is always a subset of S.

The closure operator can be defined dually and closed sets can be defined intuitionistically (without using negation) and predicatively as sets equal to their closures. The closure operator is also a saturation operator. We will briefly examine the formal definition of the saturation operator and its main properties. In the formal definition we use equality over propositions to mean equiprovability, the equality of the category CPROP.

```
definition is_saturation :=λ C:REL.λ A:Ω^C ⇒_1 Ω^C. ∀ U,V. (U ⊆ A V) =(A U ⊆ A V).
definition is_reduction :=λ C:REL.λ J: Ω^C ⇒_1 Ω^C. ∀ U,V. (J U ⊆ V) =(J U ⊆ J V).
theorem saturation_expansive: ∀ C,A. is_saturation C A → ∀ U. U ⊆ A U.
theorem saturation_monotone: ∀ C,A. is_saturation C A → ∀ U,V. U ⊆ V → A U ⊆ A V.
theorem saturation_idempotent: ∀ C,A. is_saturation C A → ∀ U. A (A U) = A U.
```

The direct proofs that concrete interior and closure are reduction and saturation operators, the definition of the formal interior and closure operators, and the direct proofs that the latter are closure and interior operators, are all given in (Sambin 2011) in the concrete setting we have presented so far. We have not formalized these direct definitions and proofs. Instead, in Section 5.3 we give them in the purely algebraic setting of overlap algebras and we will recover the concrete definitions as instances of the abstract ones.

3.5. *Basic Topologies*

A *Basic Topology* is the point-free counterpart of a Basic Pair. Intuitively, we can obtain a Basic Topology from a Basic Pair by rephrasing all properties of a Basic Pair in terms of the formal side only and then forgetting the concrete side. Since the concrete side is needed to express the forcing relation, we are also forced to drop the forcing relation. What we are left with is the topological structure over the basic opens defined by the formal interior and closure operators. It is this structure that is abstractly given in (Sambin 2011) and that we formalize here.

A Basic Topology is a set coupled with two compatible saturation and reduction operators A and J, meant to be respectively the formal interior and formal closure operators. The compatibility relation has a clear topological meaning: the interior of a set U meets (or overlap $\between$) the closure of V if and only if U already meets the closure of V.

```
record basic_topology: Type1 :={
   carrbt:> REL; A: Ω^carrbt ⇒_1 Ω^carrbt; J: Ω^carrbt ⇒_1 Ω^carrbt;
   A_is_saturation : is_saturation ? A; J_is_reduction : is_reduction ? J;
   compatibility : ∀ U,V. (A U ≬ J V) =(U ≬ J V) }.
```

A morphism of Basic Topologies from S to T is a continuous relation that is defined as a relation such that reduced sets of S are mapped existentially to reduced sets of T, and the saturated sets of T are counter-mapped universally to saturated sets of T. This definition generalizes the standard definition of continuous function in point free topology by replacing functions with relations and by describing independently the behavior on reduced and saturated sets.

```
record continuous_relation (S,T: basic_topology) : Type1 :={
   cont_rel :> S ⇒^r_1 T;
   reduced: ∀U:Ω^S. U = J ? U → cont_rel U = J ? (cont_rel U);
   saturated: ∀U. U = A ? U → cont_rel^{-*} U = A ? (cont_rel^{-*} U) }.
```

Continuous relations form a large (or 1-sized) setoid with the proper notion of equivalence, i.e. $r = s$ if and only if their existential pre-image on any point gives the same sets, once saturated. Note that, in the type of reduced, the coercion image is automatically inserted when cont_rel is applied to the set U.

```
definition continuous_relation_setoid : basic_topology → basic_topology → setoid1.
 ...λr,s:continuous_relation S T.∀b. A ? (r^- {b}) = A ? (s^- {b}) ...
definition BTop: category1. ...
```

The proof that Basic Topologies form a category is complex and requires some deep intermediate lemmas (fundamental adjunctions and fundamental symmetry). These will also later be found in an algebraic form as the properties required on O-Basic Topologies.

4. The main result

In the basic picture, Basic Topologies take the role of locales in formal topology. They are (a generalization of) the point-free representation of point-wise topological spaces. Categorically, the previous sentence is fully justified if there exists a categorical full embedding of Topological Spaces/Basic Pairs into locales/Basic Topologies. However, topological spaces do not fully embed into locales. The main result presented by Sambin, at the meeting for his 60th birthday, after the formalization presented here was completed, is that Basic Pairs fully embed into Basic Topologies instead. Our main contribution is the formalization of part of this result.

We show that a functor exists from the category of Basic Pairs to the category of Basic Topologies. The type of this functor is BP $\Rightarrow^c_2$ BTop, where $\Rightarrow^c_2$ denotes the 2-sized arrows of the very large category of large categories, i.e. the functors between large categories.

The proof given in (Sambin 2011) is a long construction made of many small lemmas, lifting the properties of Basic Pairs, defined in terms of sets and relations, to those of Basic Topologies, defined in terms of power-classes and reduction/saturation operators. Moreover, since all the definitions given so far are concrete, a first attempt at formalizing Sambin's proof in Matita failed since definitions were often unfolded too much. This meant the topological intuition was lost, and proof size exploded. Partially, this is a failure of the simplification mechanisms of Matita 1/2. The system allows the user to perform a one-step simplification operation (definition expansion or computation) or a full simplification in a user provided sub-formula (identified with mouse clicks). However, when an intermediate simplification is required, the user can only pass through intermediate expressions that may be large and uninformative; hence, topological intuition is lost.

Moreover, Sambin's proofs are all conceived at a higher, topological level. The low-level proofs were not acceptable to him as a formalization of the book.

We abandoned the fully concrete path to the proof of the main result and we made a detour into the algebraic world of *Overlap Algebras* (Ciraulo and Sambin 2010), at that time only partially studied. In the coming Sections we will formalize the fully algebraic counterparts of Basic Pairs and Basic Topologies—O-Basic Pairs and O-Basic Topologies, respectively—and we will show the existence of a functor between them. Finally, we obtain the main result in the concrete setting by instantiating the algebraic result. Interestingly, this is possible since the concrete embedding is full, even if it later turned out that the algebraic embedding is not. Our formalization prompted a search for the largest full sub-category of O-Basic Pairs that makes the embedding full. It is conjectured that O-Basic Pairs built from Ciraulo's atomic Overlap Algebras form the intended category.

5. Algebraic presentation

Intuitively, point-free topology studies lattices of the open sets of the topology. Similarly, in order to obtain an algebraic description of Basic Topologies, we need to talk algebraically about the same lattices. However, in order to assume the existence of a pair of compatible saturation and reduction operators, we need to express the compatibility condition ($\mathrm{A\ U} \between \mathrm{J\ V} = \mathrm{U} \between \mathrm{J\ V}$) algebraically using an algebraic counterpart of overlap. Classically, $\mathrm{X} \between \mathrm{Y}$ is equivalent to $\mathrm{X} \cap \mathrm{Y} \neq \emptyset$, but the latter condition is intuitionistically weaker. Since overlap hides an existential quantifier, the language of lattices is not expressive enough to talk about overlap in a complete way (in the intuitionistic sense).

This limitation is not noticed by formal topologists that work in an impredicative setting. This is because impredicatively a lattice has enough structure to rephrase most conditions on some point using the *sup* of the collection of all the points that satisfy some weaker condition (like being below it). Predicatively these collections are not sets, but proper classes. This restricts the expressiveness of the lattice language.

In Sambin's Basic Picture, all impredicative constructions are avoided. Constructions are made more effective (in the computational sense) as the overlap operator captures and hides all uses of the intuitionistic existential quantifier. In order to develop Sambin's theory algebraically, our key structure is a theory of lattices enriched with an abstract version of the overlap operator (written $\gtrless$). The structure obtained is called an *Overlap Algebra* and admits, as an example, the class of subsets of a given set. We note, however, that not all O-Algebras are obtained in this way (Ciraulo and Sambin 2010).

We formalize O-Algebras in Matita, even if it turns out that a slightly weaker structure is sufficient for our purposes.

5.1. *O-Algebras*

```
record OAlgebra : Type2 := {
  oa_P :>SET1;   oa_leq : oa_P × oa_P ⇒₁ CPROP;  oa_overlap : oa_P × oa_P ⇒₁ CPROP;
  oa_meet: ∀I:SET.(I ⇒₂ oa_P) ⇒₂ oa_P;                  oa_join : ∀I:SET.(I ⇒₂ oa_P) ⇒₂ oa_P;
 oa_leq_refl : ∀a:oa_P. a ≤ a;                oa_leq_antisym: ∀a,b:oa_P.a ≤ b → b ≤ a → a = b;
 oa_leq_trans : ∀ a,b,c:oa_P.a ≤ b → b ≤ c → a ≤ c;  oa_overlap_sym: ∀ a,b:oa_P.a ≷ b → b ≷ a;
  oa_meet_inf: ∀ I:SET.∀ p_i:I ⇒₂ oa_P.∀ p:oa_P.p ≤ (⋀ p_i) = (∀ i:I.p ≤ (p_i i));
  oa_join_sup: ∀ I:SET.∀ p_i:I ⇒₂ oa_P.∀ p:oa_P.(⋁ p_i) ≤ p = (∀ i:I.p_i i ≤ p);
  oa_one: oa_P;      oa_zero: oa_P;      oa_zero_bot: ∀ p:oa_P.𝟎 ≤ p; oa_one_top: ∀ p:oa_P.p ≤ 𝟏;
  oa_overlap_preserves_meet_: ∀ p,q:oa_P.p ≷ q → p ≷ ⋀{ x ∈ BOOL | if x then p else q |... };
 oa_join_split : ∀ I:SET.∀ p.∀ q:I ⇒₂ oa_P.p ≷ (⋁ q) = (∃ i:I.p ≷ (q i));
  oa_density: ∀ p,q.(∀ r.p ≷ r → q ≷ r) → p ≤ q }.
```

Lattice operators are infinitary. This complicates the writing of the property named oa_overlap_preserves_meet_ that indexes the meet over the set of Booleans to obtain its usual binary version. Later this record projection is wrapped inside a lemma with the more readable statement ∀ p,q:oa_P.p ≷ q → p ≷ p⋀q.

The best approximation of the notion of 'relation' between O-Algebras is obtained abstracting the four images induced by a relation on the power classes of its domain and codomain. The conditions required to link them are elegant: two adjunctions (expressed using ≤, algebrizing a universal condition) and one condition, called symmetry. Symmetry is syntactically similar to an adjunction, and can be expressed using only ≷.

```
record ORelation (P,Q : OAlgebra) : Type2 :={
  or_f_        : P ⇒₂ Q;   or_f_minus_star_ : P ⇒₂ Q;
  or_f_star_   : Q ⇒₂ P;   or_f_minus_      : Q ⇒₂ P;
  or_prop1_ : ∀ p,q. ( or_f_ p ≤ q) = (p ≤ or_f_star_ q);
  or_prop2_ : ∀ p,q. (or_f_minus_ p ≤ q) = (p ≤ or_f_minus_star_ q);
  or_prop3_ : ∀ p,q. ( or_f_ p ≷ q) = (p ≷ or_f_minus_ q) }.
```

Since O-Relations will be the arrows of the categories of Overlap Algebras, they must form a setoid with the pairwise equivalence relation.

```
definition ORelation_setoid : OAlgebra → OAlgebra → setoid2.
... ( or_f_minus_star_ ?? p = or_f_minus_star_ ?? q) ∧ (or_f_minus_ ?? p = or_f_minus_ ?? q) ∧
   ( or_f_ ?? p = or_f_ ?? q) ∧ (or_f_star_ ?? p = or_f_star_ ?? q)) ...
```

We lift every field of the O-Relation record to the morphism level for rewriting purposes. We also refrain from directly using fields that, for this reason, were named with a trailing underscore. We attach a postfix notation to all images but or_f, declaring it a coercion.

Name	Notation	Type
or_f		∀ P,Q:OAlgebra.(ORelation_setoid P Q) ⇒₂ (P ⇒₂ Q)
or_f_minus_star	"r^{-*}"	∀ P,Q:OAlgebra.(ORelation_setoid P Q) ⇒₂ (P ⇒₂ Q)
or_f_star	"r^{*}"	∀ P,Q:OAlgebra.(ORelation_setoid P Q) ⇒₂ (Q ⇒₂ P)
or_f_minus	"r^{-}"	∀ P,Q:OAlgebra.(ORelation_setoid P Q) ⇒₂ (Q ⇒₂ P)

We declare the four images in currified form, since they are always used as unary operators. Composition of O-Relations is pairwise composition in the appropriate direction.

```
definition ORelation_composition : ∀ P,Q,R.
  (ORelation_setoid P Q) × (ORelation_setoid Q R) ⇒₂ (ORelation_setoid P R).
 ...λ G,F. ...(G ∘ F) ...(G⁻* ∘ F⁻* ) ...(F* ∘ G*) ...(F⁻ ∘ G⁻) ...
```

Finally we can show that O-Algebras, together with O-Relations, form a category. The proof is routine, since the associativity and identity properties are inherited from SET.

definition OA : category2. . . .OAlgebra . . .**intros** (o o1); **apply**(ORelation_setoid o o1) . . .
notation "A $\Rightarrow^{o}_{2}$ B" := arrows2 OA A B.

The reader may have noticed that we have defined O-Algebras as having size 1 (OA is a very large category). So far, nothing prevents us from defining a smaller version. However, in the next section we will be interested in proving that relations yield examples of (1-sized) O-Algebras whose carriers are power classes of sets. Hence we restrict ourselves to the study of large O-Algebras.

5.1.1. *From concrete relations to overlap algebras*
We can define a (full and faithful) functor from concrete relations to Overlap Algebras. The proofs of the required properties of the categorical and setoid constructions over arrows are routine, but rather tedious. We must lift several properties on single elements to sets and prove the fundamental adjunctions and symmetry properties of O-Relations. This is a consequence of the properties of exchange of quantifiers in intuitionistic logic.

definition Pow: objs1 REL → OAlgebra.
intro A; . . .(Ω^A) . . .subseteq . . .overlaps . . .big_intersects . . .big_union . . .{x | True} . . .
definition orel_of_rel : ∀o1,o2:REL. o1 $\Rightarrow^{r}_{1}$ o2 → (Pow o1) $\Rightarrow^{o}_{2}$ (Pow o2).
intros(o1 o2 c); . . .(image ?? c) . . .c^{-*} . . .c^{*} . . .c^{-} . . .

Though not required for the final result, we prove that the functor is full and faithful. To show that the functor is full, we need to recover the relation given two sets A, B and an O-Relation $\langle f, f^{-*}, f^{-}, f^{*}\rangle$ between Ω^A and Ω^B. The relation is $\lambda x, y. y \in f(\{x\})$, but the proof of fullness is long yet routine.

definition POW: carr3 (REL $\Rightarrow^{c}_{3}$ OA). . . .Pow . . .orel_of_rel . . .
theorem POW_faithful: faithful2 ?? POW.
theorem POW_full: full2 ?? POW.

POW is a large functor and REL (1-sized) is implicitly coerced to the size of OA (size 2).

5.2. *Algebraic presentation of Basic Pairs*

Now that we have an algebraic version of sets and relations, in the form of O-Algebras and O-Relations, we can give the definition of the category of O-Basic Pairs by mimicking the definition of concrete Basic Pairs. Note that the algebraic category is strictly larger than the concrete one: where we used to have two sets linked by a relation, we now have two O-Algebras (intuitively, two power class lattices) and an O-Relation (made of four functions). The proofs of the required properties look the same as the ones over Basic Pairs. However, they are now phrased using larger sized objects, and we do not have to worry about definition expansion, since everything is now kept abstract. In particular, the system has less work to do during reduction and unification, especially in the case of non-convertible terms. Indeed, while several conversion strategies are usually implemented to avoid full reduction during conversion of convertible terms (see (Asperti et al. 2009a) for an example in Matita 1.0), when the system checks non convertible ones it needs, in the worst case, to fully normalize terms.

record Obasic_pair: Type2 :={ Oconcr: OA; Oform: OA; Orel: Oconcr $\Rightarrow_2^o$ Oform }.
notation "x $\Vdash$ y" := fun21 ??? (Orel ?) x y.
notation "$\Vdash$" := Orel ?.
record Orelation_pair (BP1,BP2: Obasic_pair): Type2 :={
  Oconcr_rel: (Oconcr BP1) $\Rightarrow_2^o$ (Oconcr BP2); Oform_rel: (Oform BP1) $\Rightarrow_2^o$ (Oform BP2);
  Ocommute: $\Vdash \circ$ Oconcr_rel = Oform_rel $\circ$ $\Vdash$ }.
notation "r_c" := Oconcr_rel ?? r.
notation "r_f" := Oform_rel ?? r.
definition Orelation_pair_equality : $\forall$ o1,o2. equivalence_relation2 (Orelation_pair o1 o2).
 intros(o1 o2 c); . . .(λ r,r'. $\Vdash \circ r_c = \Vdash \circ r'_c$) . . .
definition OBP: category2. . . .Obasic_pair . . .Orelation_pair_setoid . . .
notation "A $\Rightarrow_2^{obp}$ B" := arrows2 OBP A B.
notation "$\Box_t$" := fun12 ?? (or_f_minus_star ??) (Orel t)).
notation "$\Diamond_t$" := fun12 ?? (or_f ??) (Orel t).
notation "'Rest' t " := fun12 ?? (or_f_star ??) (Orel t).
notation "'Ext' t " := fun12 ?? (or_f_minus ??) (Orel x).

5.3. *Algebraic presentation of saturation and reduction operators*

The algebraic presentation of reduction and saturation is identical to the concrete one, where the category of relations REL is replaced by the category of O-Algebras.

definition is_o_saturation := λ C:OA.λ A:C $\Rightarrow_1$ C.$\forall$ U,V. (U $\leq$ A V) =(A U $\leq$ A V).
definition is_o_reduction := λ C:OA.λ J:C $\Rightarrow_1$ C.$\forall$ U,V. (J U $\leq$ V) =(J U $\leq$ J V).
theorem o_sat_expa: $\forall$ C,A. is_o_saturation C A $\rightarrow \forall$ U. U $\leq$ A U.
theorem o_sat_monot: $\forall$ C:OA.$\forall$ A:C $\Rightarrow_1$ C. is_o_saturation ? A $\rightarrow \forall$ U,V. U $\leq$ V $\rightarrow$ A U $\leq$ A V.
theorem o_sat_idemp: $\forall$ C:OA.$\forall$ A:C $\Rightarrow_1$ C. is_o_saturation ? A $\rightarrow \forall$ U. A (A U) =A U.

5.3.1. *From concrete to algebraic Basic Pairs*

Having defined both concrete and algebraic Basic Pairs, we prove the existence of a full and faithful functor between the two categories. The proofs of all required conditions are obtained by lifting the corresponding conditions on POW.

definition o_bp_of_bp: basic_pair $\rightarrow$ Obasic_pair.
 intro b . . .POW (concr b) . . .POW (form b) . . .$\text{POW}_{\Rightarrow}$ (rel b) . . .
definition o_relation_pair_of_relation_pair : $\forall$ BP1,BP2. relation_pair BP1 BP2 $\rightarrow$
  Orelation_pair (o_bp_of_bpair BP1) (o_bp_of_bp BP2).
 intros (BP1 BP2 c) . . .$\text{POW}_{\Rightarrow}$ r_c . . .$\text{POW}_{\Rightarrow}$ r_f . . .
definition BP_to_OBP: carr3 ((category2_of_category1 BP) $\Rightarrow_3^c$ OBP).

Interestingly, the proofs are heavily based on conversion: for instance, even if equality of Continuous Relations and O-Continuous Relations are defined in terms of different entities (one with relations on small sets, the other with four functions on large classes), when an O-Continuous Relation is obtained from a Continuous Relation the two equalities yield the same conditions by conversion. Indeed, in a logic that does not internalize conversion in its rules, we believe that all our functors from concrete to algebraic categories would be more tedious to define.

```
theorem BP_to_OBP_faithful: faithful2 ?? BP_to_OBP.
theorem BP_to_OBP_full: full2 ?? BP_to_OBP.
```

5.4. *Algebraic presentation of Basic Topologies*

We may easily define the algebraic versions of saturation and reduction operators over O-Algebras. We then define O-Basic Topologies by mimicking the definition of Basic Topologies in the algebraic setting. Note, how we need to use the abstract overlap operator of O-Algebras to express compatibility.

```
record Obasic_topology: Type2 :={
   Ocarrbt:> OA; oA: Ocarrbt ⇒_2 Ocarrbt; oJ: Ocarrbt ⇒_2 Ocarrbt;
   oA_is_saturation : is_o_saturation ? oA; oJ_is_reduction : is_o_reduction ? oJ;
   Ocompatibility: ∀U,V. (oA U ≬ oJ V) =(U ≬ oJ V) }.
record Ocontinuous_relation (S,T: Obasic_topology) : Type2 :={
   Ocont_rel:> S ⇒_2^o T;
   Oreduced: ∀U:S. U = oJ ? U → Ocont_rel U =oJ ? (Ocont_rel U);
   Osaturated: ∀U:S. U = oA ? U → Ocont_rel^{-*} U =oA ? (Ocont_rel^{-*} U) }.
definition OBTop: category2. ...Obasic_topology ...Ocontinuous_relation_setoid ...
notation "A ⇒_2^{obt} B" := arrows2 OBTop A B.
```

5.4.1. *From concrete to algebraic Basic Topologies*

The full and faithful functor from concrete to algebraic topologies is obtained in a similar fashion to the one from concrete to algebraic Basic Pairs. This is achieved by mapping all relations to the corresponding O-Relation, and all saturation and reduction operator to the corresponding algebraic one. Reduction is also heavily used in the various proofs to recover the concrete definition from the abstract ones. In particular, the functor is full essentially because POW is, therefore we can recover all necessary relations.

```
definition obtop_of_btop: basic_topology → Obasic_topology.
 intro (b) ...(POW b) ...(A b) ...(J b) ...
 ...(A_is_saturation b) ... ( J_is_reduction b) ...(compatibility b) ...
definition ocrel_of_crel: ∀BT1,BT2.continuous_relation BT1 BT2 →
  Ocontinuous_relation (obtop_of_btop BT1) (obtop_of_btop BT2).
 intros (BT1 BT2 c); constructor 1;
  [ apply ( orelation_of_relation ?? c) | apply (reduced ?? c) | apply (saturated ?? c) ]
definition BTop_to_OBTop: carr3 ((category2_of_category1 BTop) ⇒_3^c OBTop).
theorem BTop_to_OBTop_faithful: faithful2 ?? BTop_to_OBTop.
theorem BTop_to_OBTop_full: full2 ?? BTop_to_OBTop.
```

6. The proof of the main theorem

We first prove our main result in the algebraic setting and later use this as a lemma for the concrete case.

6.1. *Algebraic setting*

The algebraic proof follows Sambin's concrete proof quite closely, since Sambin's proof is essentially algebraic.

The functor from O-Basic Pairs to O-Basic Topologies needs to map both objects and arrows. The latter are simply mapped by forgetting the concrete side and keeping only the formal side. Thus the non-trivial part of the construction is obtaining the reduction and saturation operators (corresponding to the formal closure and interior operators). Sambin observed that the definition of the interior and closure operators (both on the concrete and formal sides) can be obtained by simply composing in sequence a universal image and an existential counter-image or *vice versa*. Thus, we obtain the two operators as a simple composition of the four components of our O-Relation. Proving that these operators satisfy the required properties relies on several lemmas that have been mostly proved via diagram chasing of some sort. This is because almost all the hypotheses and definitions are equations in the algebraic setting. The functor obtained is called OR.

```
definition o_bt_of_o_bp: OBP → OBTop.
 intro t ...(Oform t) ...(□ₜ ∘  Extt ) ... (◇ₜ ∘  Restt ) ...
 ...  change with ((◇ₜ (⊩* V) ⋈⊩⁻*(⊩⁻ U)) = (U ⋈(◇ₜ (⊩* V))));
      apply (.= (or_prop3 ?? (⊩) (⊩* V) ?));   apply (.= #‡lemma_10_3_a); ...
definition  o_continuous_relation_of_o_relation_pair :
 ∀BP1,BP2.BP1 ⇒₂^obp BP2 → (o_bt_of_o_bp BP1) ⇒₂^obt (o_bt_of_o_bp BP2).
 intros (BP1 BP2 t); ...(t_f) ...
definition OR : carr3 (OBP ⇒₃^c OBTop).
 ...o_bt_of_o_bp...o_continuous_relation_of_o_relation_pair ...
```

The functor OR is not full. In the algebraic setting there is not enough information on the formal side to recover the concrete part of a continuous relation from the abstract part. It was later discovered that fullness holds in the concrete setting and it is currently conjectured that this holds for every atomic O-Basic Topology.

6.2. *Concrete setting*

We use conversion to prove the main theorem in the concrete setting, exploiting the abstract version. The action of the functor on objects and morphisms follows:

```
definition btop_of_bp: basic_pair → basic_topology.
 intro bp; letin obt :=(OR (BP_to_OBP bp));
 constructor 1;
  [apply (form bp); | apply (oA obt); | apply (oJ obt);
  |apply (oA_is_saturation obt);|apply (oJ_is_reduction obt );| apply (Ocompatibility obt); ]
qed.
```

The first line of the script is the main proof step: we first embed the Basic Pair bp into the algebraic world of O-Basic Pairs, obtaining an O-Basic Pair. We then apply the functor OR to build the O-Basic Topology obt. At this point we do not have any functor or general construction to obtain a Basic Topology from an O-Basic Topology. However, we know that its carrier must be (form bp). The request to provide the saturation and reduction operators remains, along with a proof that they form a Basic Topology. We recover all these fields from obt, leaving the burden of verifying that they have the expected types to the system, using only conversion.

A priori, nothing grants this property, that is a highly non-trivial consequence of the deep correspondence between the algebraic and concrete definitions. Moreover, we are relying on the fact that by reducing a large type we can obtain a small one. For instance, to provide the saturation operation, we inhabit the goal $\Omega^{\texttt{form bp}} \Rightarrow_1 \Omega^{\texttt{form bp}}$ with (oA obt) that has the larger type obt $\Rightarrow_2$ obt.

The action of the functor on continuous relations is obtained in the same way.

```
definition continuous_relation_of_relation_pair :
 ∀BP1,BP2.relation_pair BP1 BP2 → continuous_relation (btop_of_bp BP1) (btop_of_bp BP2).
 intros (BP1 BP2 rp); constructor 1; letin ocr :=(OR⇒ (BP_to_OBP⇒ rp));
  [ apply (rp_f); | apply (Oreduced ?? ocr); | apply (Osaturated ?? ocr); ]
qed.
```

Our main result follows: there exists a functor from Basic Pairs to Basic Topologies.

7. Conclusions and future works

The formalization described here may be found on-line at `http://matita.cs.unibo.it/library/formal_topology/`. It is self contained (from the definition of predicative logical connectives). It consists of 17 files with a total of 3721 lines and 322 definitions, lemmas and theorems. We do not distinguish between definitions and lemmas, since most definitions are of setoids, morphisms, categories and functors, and require proofs of properties that, in several cases, are non-trivial. We formalized merely a small percentage of (Sambin 2011). However, we formalized material from throughout the entire book.

Overlap Algebras are to intuitionistic subset theory what Boolean algebras are to classical subset theory. They enrich Heyting algebras with a primitive overlap operator that has the semantics of the intuitionistic existential quantifier. This permits us to recast existential formulae in term of overlaps of certain sets.

Our formalization is the first convincing proof that constructive mathematics can be recast and algebraized in terms of Overlap Algebras to avoid quantifiers in formulae, yielding material with a categorical flavour, where diagram chasing forms an important style of proof. Formalization in this style has several advantages when working in an interactive theorem prover. From the point of view of the user, proofs are done at a higher level of detail, making them more general, intuitive and understandable, reducing the search space. From the point of view of automation, the most efficient techniques over equational theories (like superposition) are more efficacious than those for general deduction; From the point of view of type checking and script execution, the algebraic approach avoids potentially expensive reductions by replacing defined entities with axiomatized ones, potentially greatly improving the speed of the conversion check.

Our formalization suggested several changes to the logical framework and implementation of Matita. In particular, the design and management of universes, the management of coercions, and the possibility of driving the unification algorithm of Matita to automatically solve unification problems that require automatic enrichment of mathematical structures have all been changed due to the formalization in this paper. The next major release of Matita will provide all these enhancements. In this paper we have described some sub-optimal solutions that are likely to work for similar proof assistants (such as Coq) without modification.

We plan to extend the superposition based automatic theorem prover of Matita (Asperti and Tassi 2010) in order to deal with setoid rewriting. We also plan to tune the system to understand how much of the present formalization can be carried out autonomously by Matita. This has not been done yet since Sambin was interested in verifying the proofs of his book, and not the provability of the statements. In particular, it is worth noting that, excluding the technical commands that declare notations, coercions and similar things, the extrinsic de Bruijn factor of the formalization is about 1. This reflects an unusual level of accuracy of (Sambin 2011).

Theoretical and practical work is still needed to cope with an explicit hierarchy of universes without duplicating effort. A first solution we plan to experiment with is modifying the system library to generate lifted versions of parts of the library on-the-fly.

Finally, we aim to exploit Matita's new features, like unification hints for setoid rewriting and non uniform coercions to currify setoid morphisms to formalize the convergent side of the Basic Picture, along with the important class of inductively generated formal topologies (see (Coquand et al. 2003; Valentini 2005)). The latter are special instances of Basic Topologies where the saturation and reduction operators are independently given as the inductive and co-inductive closures of some sets of covering axioms respectively. Compared to the completely abstract Basic Topologies, they are characterized by a greater degree of computability. This should allow one to extract programs working on concrete computational data types (like computable Cauchy Reals) from proofs about more abstract data (like general Dedekind Reals). They also have an important role in formal topology, since several constructions that fail in the general case (like Tychonoff's theorem) can be carried out in the inductively generated case. Interestingly, for us, is the fact that they complicate the formalization by mixing together abstract algebraic reasoning with induction and co-induction. We expect them to be the next important driver for the development of Matita, in particular with respect to automation.

We would like to thank Giovanni Sambin and Milly Maietti for introducing us to the Basic Picture and for useful discussions on constructive mathematics and Minimal Logic.

References

Asperti, A. and Armentano, C. (2008) A page in number theory. Journal of formalized reasoning, Volume 1, Pages 1–23.

Asperti, A. and Tassi, E. (2010) Smart matching. Proceedings of MKM2010, LNCS, Volume 6167, Pages 263–277.

Asperti, A., Sacerdoti Coen, C., Tassi, E. and Zacchiroli, S. (2007) User interaction with the Matita proof assistant. Journal of automated reasoning, Volume 39(2), Pages 109–139.

Asperti, A., Ricciotti, W., Sacerdoti Coen, C. and Tassi, E. (2009a) A compact kernel for the Calculus of Inductive Constructions. Journal Sadhana, Volume 34(1), Pages 71–144.

Asperti, A., Ricciotti, W., Sacerdoti Coen, C. and Tassi, E. (2009b) Hints in unification. Proceedings of TPHOLs2009, LNCS, Volume 5674, Pages 84–98.

Ciraulo, F. and Sambin, G. (2010) The overlap algebra of regular opens. Journal of pure and applied algebra, Volume 214, Pages 1988–1995.

Coquand, T. and Huet, G. P. (1988) The calculus of constructions. Information and Computation, Volume 76(2/3), Pages 95–120.

Coquand, T., Sambin, G., Smith, J. M. and Valentini, S. (2003) Inductively generated formal topologies. Annals of pure and applied logic, Volume 124(1-3), Pages 71–106.

Courant, J. (2002) Explicit universes for the calculus of constructions. Proceedings of TPHOLs2002, LNCS, Volume 2410, Pages 115–130.

Cruz-Filipe, L., Geuvers, H. and Wiedijk, F. (2004) C-CoRN, the constructive Coq repository at Nijmegen. Proceedings of MKM2004, LNCS, Volume 3119, Pages 88–103.

Cubric, D., Dybier, P. and Scott, P. (1997) Normalization and the yoneda embedding. Mathematical structures in computer science, Volume 8(2), Pages 153–192.

Johnstone, P. T. (1983) The point of pointless topology. Bullettin of the american mathematical society, Volume 8, Pages 41–53.

Luo, Z. (1989) ECC, an extended calculus of constructions. Proceedings of LICS1989 , IEEE Press, Pages 386–395

Maietti, M. E. and Sambin, G. (2005) From sets and types to topology and analysis, toward a minimalist foundation for constructive mathematics. Oxford University Press. Chap. 6.

Padovani, L. and Zacchiroli, S. (2006). From notation to semantics: There and back again. Proceedings of MKM2006, LNAI, Volume 4108, Pages 194–207.

Paulin-Mohring, C. (1996) Définitions inductives en théorie des types d'ordre supŕieur. Habilitation à diriger les recherches, Université Claude Bernard Lyon I.

Sacerdoti Coen, C. (2006) A semi-reflexive tactic for (sub-)equational reasoning. Proceedings of TYPES2006, Volume 3839, Pages 98–114.

Sacerdoti Coen, C. and Tassi, E. (2010) Nonuniform coercions via unification hints. To appear in the proceedings of TYPES 2009, EPTCS, ISSN: 2075-2180, DOI: 10.4204/EPTCS.

Sacerdoti Coen, C. and Tassi, E. (2008) A constructive and formal proof of Lebesgue's dominated convergence theorem in the interactive theorem prover Matita. Journal of formalized reasoning, Volume 1, Pages 51–89.

Sacerdoti Coen, C. and Zacchiroli, S. (2008) Spurious disambiguation errors and how to get rid of them. Journal of mathematics in computer science, Volume 2, Pages 355–378.

Sambin, G. To be published in (2011) The Basic Picture: a structural basis for constructive topology. Oxford University Press.

Sozeau, M. (2009) A new look at generalized rewriting in type theory. Journal of formalized reasoning, Volume 2(1), Pages 41–62.

Sozeau, M. and Oury, N. (2008) First-class type classes. Proceedings of TPHOLs2008, LNCS, Volume 5170, Pages 278–293.

Valentini, S. (2005) The problem of the formalization of constructive topology. Archive for mathematical logic, Volume 44(1), Pages 115–129.

Werner, B. (1997) Sets in types, types in sets. Proceedings of TACS97, LNCS, Volume 1281, Pages 530–546.